Homework 5

Benjamin Anderson II

```
library(tidyverse)
library(knitr)
```

Tasks that require an answer are bolded (inside ** in the .qmd file). For any task that includes a question (i.e. it ends with "?"), you should also answer the question in sentence form.

Vectors and Vector Functions

1.

(1 pt)

The following three chunks are attempts to create vectors, but each one has a problem. Either the vector created is of the wrong type, or there is a syntax error. **Identify the problem, then fix the code in each chunk.**

This should be a logical vector of length 5:

The issue with the code is that the elements of the vector are in quotation marks

```
# c("TRUE", "FALSE", "TRUE", "TRUE", "FALSE") # BAD
c(TRUE, FALSE, TRUE, TRUE, FALSE)             # GOOD
```

```
[1]  TRUE FALSE  TRUE  TRUE FALSE
```

This should be a character vector of length 4:

The issue with the code is that it has no quotation mark after the word "carrot"

```
# c("potato", "carrot, "eggplant", "lettuce") # BAD
c("potato", "carrot", "eggplant", "lettuce")  # GOOD
```

```
[1] "potato"   "carrot"   "eggplant" "lettuce"
```

This should be a double vector of length 4:

The issue with the code is that there is no comma between 1.1 and 6.4.

```
# c(1.1 6.4, 1.5, 0.9) # BAD
c(1.1, 6.4, 1.5, 0.9)  # GOOD
```

```
[1] 1.1 6.4 1.5 0.9
```

2.

(2 pts)

Consider the vector `x`:

```
x <- c("10", "100%", "$1000")
```

What type of vector is `x`?

```
typeof(x)
```

```
[1] "character"
```

x is a "character" vector

I mentioned the dangers of coercion with `as.numeric()`. readr, a tidyverse package, provides the function `parse_number()`. **Apply both `as.numeric()` and `parse_number()` to `x`, then in your own words describe the difference in their behaviour.**

```
as.numeric(x)
```

```
[1] 10 NA NA
```

```
parse_number(x)
```

```
[1]   10  100 1000
```

Since the second two elements of `x` are not strictly numbers the `as.numeric()` function is unable to cast them directly into numbers ("%" and "$" are not numbers, so they do not properly coerce to numbers). The `parse_numer()` function, on the other hand, does not attempt to strictly convert the element into a number, but rather parse whatever number may be in the "character" variable. *Example:*

```
parse_number(c('asdf132as', '12', 'fhgi0'))
```

```
[1] 132  12   0
```

3.

(1 pt)

Consider the following code and output:

```
x <- c(1, 2, 3, 4)
y <- c(TRUE, FALSE)
x * y
```

```
[1] 1 0 3 0
```

In your own words, describe how R arrives at the output.

Firstly, logical variables are just a typecast of integers in R, `FALSE = 0`, and `TRUE = 1`. After this, it can be inferred that the infix `*` operator is overloaded to be able to multiply the elements of vectors by looping through larger one and cycling through the smaller one. An example of how this may be implemented is shown below:

```
# No checks are done to test whether x or y are capable of this operation
out <- 1:max(x,y)
for(i in out){
  # The `- 1` and `+ 1` are added because R begins indexing at 1, not 0
  out[i] <- x[((i - 1) %% length(x)) + 1] * y[((i - 1) %% length(y)) + 1]
}
out
```

```
[1] 1 0 3 0
```

4.

(2 pts)

Consider the `starwars` dataset from dplyr.

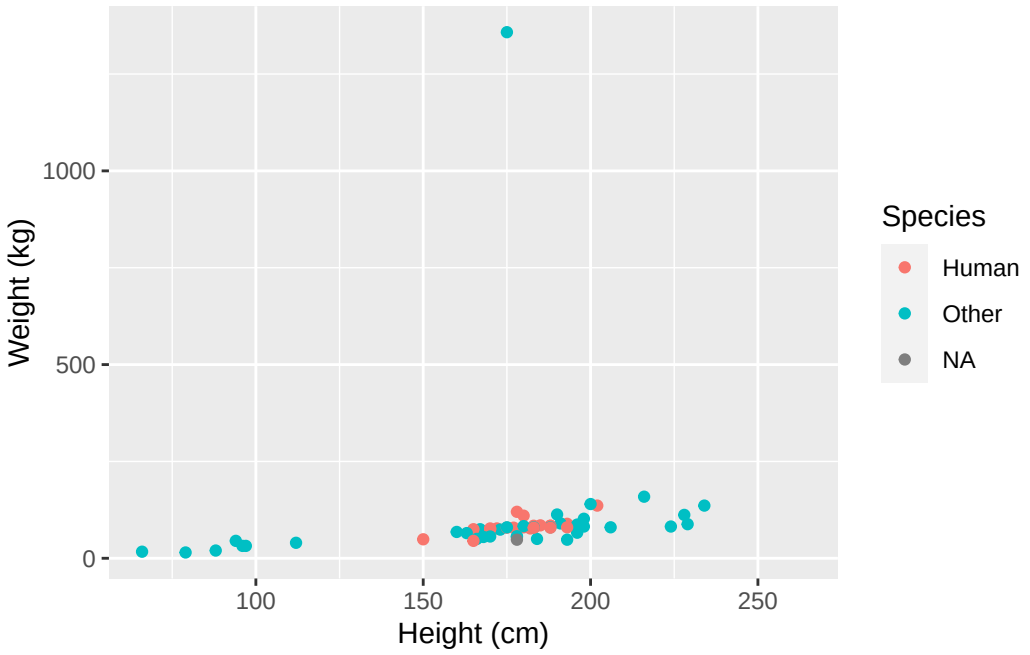
Add a column called `human` to `starwars` that takes the value `"Human"` if `species` is `"Human"` and `"Other"` otherwise.

```
starwars <- starwars |>
  mutate(human = ifelse(species == "Human", "Human", "Other"))
```

Create a scatterplot of `height` versus `mass` with points colored by your new `human` column.

```
#Really want to cut out Jabba because he's clearly an outlier, but I won't

starwars |>
  ggplot(mapping = aes(x = height, y = mass)) +
  geom_point(aes(color = human)) +
  labs(x = "Height (cm)",
       y = "Weight (kg)",
       color = "Species")
```



5.

(2 pts)

How many characters in `starwars` have more than one skin color?

Complete the following steps to answer the question.

One strategy to look for multiple skin colors, is to look to see if the value for `skin_color` contains a comma. E.g.

```
example_skin <- c("fair", "gold", "white, blue")
str_detect(example_skin, ",")
```

```
[1] FALSE FALSE  TRUE
```

Create a new column in `starwars` called `many_cols` that contains `TRUE` if the characters `skin_color` contains a comma and `FALSE` otherwise.

```
starwars <- starwars |>
  mutate(many_cols = ifelse(str_detect(skin_color, ","), TRUE, FALSE))
```

Filter `starwars` using the column `many_cols`.

```
starwars |>
  filter(many_cols) |>
  nrow()
```

```
[1] 14
```

Using the result from above, answer the question, how many characters in `starwars` have more than one skin color?

Using the nrow function we can see that there are 14 characters in the starwars dataset that have more than one skin color.

6.

Here's a small example of taking a vector that contains years and converting it to a character vector representing decades:

```r
year <- c(1900, 1901, 1909, 1910, 1921, 1931, 2001)
floor(year / 10) |> paste0("0's")
```

```
[1] "1900's" "1900's" "1900's" "1910's" "1920's" "1930's" "2000's"
```

Which functions in the second line of code are vector functions?

Here's some randomly generated data relating to prices over time:

```r
set.seed(2484) # so you all get the same "random" data
prices <- tibble(
  year = 1900:1950,
  price = rnorm(n = length(year), mean = year/10)
)
prices
```

```
# A tibble: 51 x 2
    year price
   <int> <dbl>
 1  1900  191.
 2  1901  190.
 3  1902  190.
 4  1903  191.
 5  1904  191.
 6  1905  190.
 7  1906  192.
 8  1907  187.
 9  1908  191.
10  1909  192.
# i 41 more rows
```

Add a column `decade` that is a character string representing the decade corresponding the `year`

```r
prices <- prices |>
  mutate(decade = floor(year / 10) |> paste0("0's"))
```

Use your new `decade` column to produce a summary with the mean price per decade.

```
prices |> group_by(decade) |>
  summarise(mean_price = mean(price)) |>
  kable()
```

decade	mean_price
1900's	190.3645
1910's	191.3653
1920's	192.1295
1930's	193.3587
1940's	194.5131
1950's	195.5229